Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Shorter hash-based signatures

Geovandro C.C.F. Pereira[1],*, Cassius Puodzius[1], Paulo S.L.M. Barreto[1]

Departamento de Engenharia de Computação e Sistemas Digitais (PCS), Escola Politécnica, Universidade de São Paulo. Av. Prof. Luciano Gualberto, trav. 3, 158 05508–900 São Paulo (SP), Brazil

ARTICLE INFO

Article history:
Received 2 October 2014
Revised 31 May 2015
Accepted 1 July 2015
Available online 10 July 2015

Keywords:
Embedded security
Hash-based signatures
Internet of things

ABSTRACT

We describe an efficient hash-based signature scheme that yields shorter signatures than the state of the art. Signing and verification are faster as well, and the overall scheme is suitable for constrained platforms typical of the Internet of Things. We describe an efficient implementation of our improved scheme and show memory, time, and energy consumption benchmarks over a real device, i.e. the ATmega128l 8-bit AVR microcontroller embedded in MICAz, a typical sensor node used in wireless sensor networks.

1. Introduction

Internet of Things (IoT) purports to connect a vast range of equipments via the Internet, as long as the underlying processor is large enough to support the TCP/IP protocol. This includes extremely constrained platforms, with as little as 64 KiB ROM and 4 KiB RAM as certain SIM cards. While this is commonly enough for symmetric primitives (hash functions, block and stream ciphers, and even richer constructions like authenticated encryption with associated data), it may push most asymmetric primitives beyond the available computational resources on such platforms. Yet, securing a typical Internet of Things scenario requires, at the very least, a basic public-key infrastructure (PKI) able to support public-key encryption and digital signatures, which are themselves based on asymmetric primitives.

While encryption can be attained with fairly modest resources by adopting lattice-based schemes (Hoffstein et al., 1998) or code-based schemes (Misoczki et al., 2013), offering even the most basic functionality of digital signatures on the most stringent platforms is no easy task. Obvious and more exotic candidates alike suffer from the extreme lack of computational resources on some of those platforms, which currently seem to be at, or already beyond, the bare minimum needed to establish a full-fledged PKI. This lack of an efficient signature functionality constitutes a serious hindrance to the very concept of the IoT, especially if resorting to more expensive processors or coprocessors is not an option.

Hash-based signatures, which originally appeared somewhat too far-fetched for actual deployment, turned out to be a very promising tool for the aforementioned scenario. On the one hand, their main drawback – which was a very long key generation time, have been for the most part successfully addressed in recent research works (Buchmann et al., 2007). On the other hand, practical considerations like the actual signature size and consequent bandwidth occupation, as well as leakage-resilience, have also been addressed, with very promising results (Buchmann et al., 2011b; Eisenbarth et al., 2013b; Hülsing, 2013; Rohde et al., 2008). Although they do constitute true digital signatures in the sense of public-key cryptosystems, such schemes are based on entirely symmetric primitives, which are readily available on constrained platforms, are typically very efficient, and appear to resist attacks mounted even with the help of quantum computers, to the extent that hash-based signatures have been promoted to the category of quantum-resistant, or post-quantum, cryptosystems. Yet, given the extreme scarcity of resources one finds in IoT processors, full establishment of a secure environment for realistic applications requires that all cryptographic features be made as lightweight as possible, and hash-based signatures are no exception. It therefore makes sense to look for the most efficient constructions rather than sticking with proofs of concept.

Our contribution in this paper is an efficient hash-based signature scheme that not only yields shorter signatures than the previous state of the art, but also enables faster signature generation and verification for the same security level and word size parameters. We also provide a practical implementation for a very constrained microcontroller, the ATmega128l (@7.37MHz, 4KiB SRAM and 128KiB ROM), and argue that the resulting scheme is very suitable for constrained

* Corresponding author. Tel.: +55 11 3091 9759.
E-mail addresses: geovandro@larc.usp.br (G.C.C.F. Pereira), cpuodzius@larc.usp.br (C. Puodzius), pbarreto@larc.usp.br (P.S.L.M. Barreto).

[1] The authors are supported by Intel Research grant “Energy-efficient Security for SoC Devices – Asymmetric Cryptography for Embedded Systems” 2012. P. Barreto is also supported by CNPq research productivity grant 306935/2012-0.

http://dx.doi.org/10.1016/j.jss.2015.07.007
0164-1212/© 2015 Elsevier Inc. All rights reserved.

platforms typical of the IoT, as well as similar or related scenarios, like wireless sensor networks and intelligent habitats and environments.

The remainder of this document is organized as follows. In Section 2 we introduce the essential concepts behind digital signatures in general, and hash-based signatures in particular. We describe our proposal in Section 3, assess it and provide comparisons in Section 4, evaluate its security in Section 5 and describe our implementation results and benchmarks in Section 6. We conclude in Section 7.

2. Preliminaries

In this section we first recapitulate the Winternitz one-time signatures which combined with the Merkle tree construction allows to obtain a multi-time signature scheme. We also give a brief description of the Merkle signature scheme (MSS).

2.1. Winternitz one-time signatures

The Winternitz one-time signature (W-OTS) scheme views the message representative to be signed as a sequence of L w-bit words (or chunks), denoted by $m = (m_0, \dots, m_{L-1})$, where m_i stands for an integer value in the range $0 \dots 2^w - 1$. The signature component for any particular such word m_i will be an m_ith iterated preimage of some (public) L-word hash value univocally associated to the ith component of the message representative.

Formally, let $G : \{0, 1\}^* \to \{0, 1\}^{2n}$ be a collision-resistant hash function and $F : \{0, 1\}^* \to \{0, 1\}^n$ a one-way hash function. Also let $F^k := F \circ F \circ \cdots \circ F$ be F iterated k times.

Message representative preparation:

Let $data \in \{0, 1\}^*$ denote the original document to be signed. First compute $M = (m_0, \dots, m_{\ell_1 - 1}) := G(data)$, where $\ell_1 = \lceil 2n/w \rceil$. Then, compute the checksum $CS := \sum_{i=0}^{\ell_1 - 1} (2^w - 1 - m_i)$ which is then split into w-bit words $(m_{\ell_1}, \dots, m_{\ell_1 + \ell_2 - 1})$ and appended to M, resulting the message representative $m = M || CS = (m_0, \dots, m_{L-1})$. The maximum checksum integer value is $(2^w - 1)\ell_1$, which fits in $\ell_2 = \lceil (lg((2^w - 1)\ell_1)/w \rceil$ w-bit words, therefore the total number of w-bit words is $L = \ell_1 + \ell_2$.

The triple of algorithms that defines this scheme is:

- Gen: Choose L strings $s_i \xleftarrow{\$} \{0, 1\}^n$ uniformly at random, and compute $v_i \leftarrow F^{2^w - 1}(s_i)$, for $i = 0, \dots, L - 1$. The private key is the sequence $(s_0, \dots, s_{L-1})$, and the public key is $v = F(v_0 || \cdots || v_{L-1}) \in \{0, 1\}^n$.
- Sig: To sign a message representative $m = (m_0, \dots, m_{L-1})$, compute $S_i \leftarrow F^{2^w - 1 - m_i}(s_i)$, and let the signature be the sequence of resulting values $S = (S_0, \dots, S_{L-1}) \in (\{0, 1\}^n)^L$. Notice that S_i is an m_ith iterated preimage of v_i for all $i = 0, \dots, L - 1$.
- Ver: To verify the signature $S = (S_0, \dots, S_{L-1})$ of the message representative $m = (m_0, \dots, m_{L-1})$, compute $t_i = F^{m_i}(S_i)$ for $i = 0, \dots, L - 1$ and check if $v = F(t_0 || \cdots || t_{L-1})$.

An obvious improvement to the scheme is to adopt a short secret string $s \in \{0, 1\}^n$ as the private key, and then compute either $s_i \leftarrow F(s||i)$ on demand to reducing memory usage, or else, given a longer hash K: $\{0, 1\}^n \to (\{0, 1\}^n)^L$ (say, a cryptographic sponge Bertoni et al., 2007), set $(s_0, \dots, s_{L-1}) \leftarrow K(s)$. The first approach is more adequate for memory and speed limited devices since it can be built from a block cipher-based hash function. Usually many embedded devices provide fast SW and sometimes HW AES implementations, so it can be reused for the hash construction. Given the 4KiB SRAM memory and 7.37 MHz budget available in the ATmega128l microcontroller, our implementation focus on the first approach.

2.2. Merkle signature scheme

The Merkle tree-based signature scheme of height h (defined as the distance between the root and the leaves of the tree, so as to have 2^h leaves and $2^{h+1} - 1$ total nodes) and a hash function F extends a one-time signature scheme to 2^h signable messages for each Merkle public key.

The technique consists of generating 2^h one-time key pairs $(s^{(j)}, v^{(j)})$, $0 \le j < 2^h$, for a given one-time signature scheme, and then computing a tree of hash values $q_1, \dots, q_{2^{h+1}-1}$ so that $q_i = F(q_{2i} \mid\mid q_{2i+1})$ for $1 \le i < 2^h$, and $q_{2^h + j} = F(v^{(j)})$ for $0 \le j < 2^h$. The overall public key for the scheme is $Y = q_1$.

Given the one-time signature $S^{(j)}$ verifiable under the public key $v^{(j)}$, the Merkle technique assembles an *authentication path* consisting of the tree nodes whose values are not directly computable from $v^{(j)}$ alone, but are nevertheless needed to compute the values of the parent nodes leading from $v^{(j)}$ to the root Y.

Thus, a Merkle signature is a triple $\Sigma^{(j)} = (S^{(j)}, v^{(j)}, Q^{(j)})$ where $Q^{(j)}$ is the sequence of values $(q_{\lfloor j/2^u \rfloor \oplus 1} \mid u = 0, \dots, h - 1)$ along the authentication path. The Merkle signature length is $z = |S^{(j)}| + |v^{(j)}| + h|q_i|$.

Merkle's construction allows the root value to be computed from an as yet unused (and publicly unknown) $v^{(j)}$, and then compared to Y. This ensures that $v^{(j)}$ is itself authentic, whereby $S^{(j)}$ can be verified as well.

The Merkle–Winternitz scheme, which combines Winternitz one-time signatures with an overall Merkle tree scheme, yields one of the most efficient hash-based signatures. For a Merkle–Winternitz scheme, the signature size is $z = |S^{(j)}| + |v^{(j)}| + h|q_i| = Ln + n + hn = n(L + 1 + h)$ bits.

3. Proposed scheme

Our scheme is related to that by Rohde et al. (2008), which we call here REDBP scheme for short. A small conceptual difference is responsible for our reduced signature size and higher processing speeds attainable. Furthermore, we still avail ourselves of the remarks in Eisenbarth et al. (2013b) to reduce the number of leaf generation to improve time performance.

Key generation requires heavy computational load, since all nodes up to the root must be calculated. Furthermore, many nodes are stored during this process for the setup of REDBP. Therefore, if key generation and signature calculation do not have to take place in the same device, a better memory usage can be achieved by storing some fixed nodes in ROM instead of RAM. Due to this fact, it is more suitable to perform key generation on standard PCs than on a microcontroller and enhance the setup of the signing device.

Specifically, the REDBP scheme adopts a hash function $G : \{0, 1\}^* \to \{0, 1\}^{2n}$ to create digests of the form $M = (m_0, \dots, m_{\ell_1 - 1}) = G(data)$, $m_i \in \{0, \dots, 2^w - 1\}$ and produces message representatives of the form $M||CS = (m_0, \dots, m_{L-1})$ as described in Section 2.1, with a straightforward Merkle tree construction on top of Winternitz signatures. For security level roughly 2^n (whereby forging existentially a signature takes about 2^n computational steps), that scheme sets $\ell_1 w = 2n$. Thus the G hash size is twice that of the F hash size. Since each Winternitz signature has length $|S^{(j)}| = nL \approx n\ell_1$ (omitting the checksum size ℓ_2), as a result each Merkle–Winternitz signature has length $z = n(L + 1 + h) \approx n(\ell_1 + 1 + h) = n(2n/w + 1 + h)$.

Intuitively, this is necessary to prevent precomputed collision attacks. Indeed, since only the message *data* is fed into the hash function G, an attacker could mount a Yuval-style attack (Yuval, 1979), preparing beforehand two sets of semantically equivalent messages, the first favorable to the signer and the second unfavorable, and looking for a collision between a favorable message *data* and an unfavorable one *data*$'$, finally presenting *data* to the signer and *data*$'$ to an